Hackathon UAB

School of Engineering

18th-19th May 2024

GenAI based Python Programming Assistant Prototype

Aleix Sancho Jiménez - B.S. in Mathematics & B.S. in Computer Science

Arnau Perich Iglesias - B.S. in Mathematics

Eric Jiménez Barril - B.S. in Mathematics - Junior Data Scientist

Joan González Martínez - B.S. in Mathematics & B.S. in Physics

Abstract

This report presents the development and evaluation of a prototype for a Python programming assistant application. The app is designed to assist programmers by providing real-time code suggestions, error detection, debugging assistance, and answers to general Python-related questions. Users can interact with the app through text or voice messages. This report covers the objectives, design, implementation, preliminary results and future steps of the prototype. We emphasize that the development process was constrained to a 24-hour period, during which we created the prototype, wrote the paper, and prepared the presentation.

Contents

1 Introduction

1.1 The Challenge

The format of the hackathon involved selecting a challenge from one of the three presenting companies: *Caixa d'Enginyers*, *FGC*, and *NTT Data*, and solving it within 24 hours. We chose the challenge presented by NTT Data, which was described as follows:

> "The challenge of the hackathon is to identify and utilize GenAI-based tools in different phases of software development (design, construction, testing, documentation), identifying specific benefits they bring (efficiency, agility, user experience), and reflecting on the profound transformation that traditional software development is currently undergoing, where AI can be perceived as a threat or an ally."

1.2 Our approach

Given the challenge we chose, we decided to develop an application from scratch that is able to read any code file with the extension *.py* the user selects and then open a chat window where the user can ask any questions related to their code or Python in general.

1.3 Objectives

- Implement a simple and effective Python code assistance tool based on Generative AI.
- Minimize the time from when a user has an issue to when our model resolves it, making it shorter than the time it would take the user to get the answer using tools like `ChatGPT`.
- Make sure the solution is viable both computationally and economically.
- Allow the user to interact with the program via either text or audio.

2 Design and Implementation

2.1 System Architecture

In this section the overall architecture of the assistant is described, together with the different components our application has and how they interact. Note that all the Large Language Models used are from `OpenAI`, thus we will need to access the models through the API [1]. These models will be accessed through the `Langchain` library [2].

In the Figure 1, we can see a flowchart of our app. The flow direction is always from left to right, except when we reach the final output of the model, which is then fed back as input to one of the initial nodes to ensure the model remembers the entire conversation.

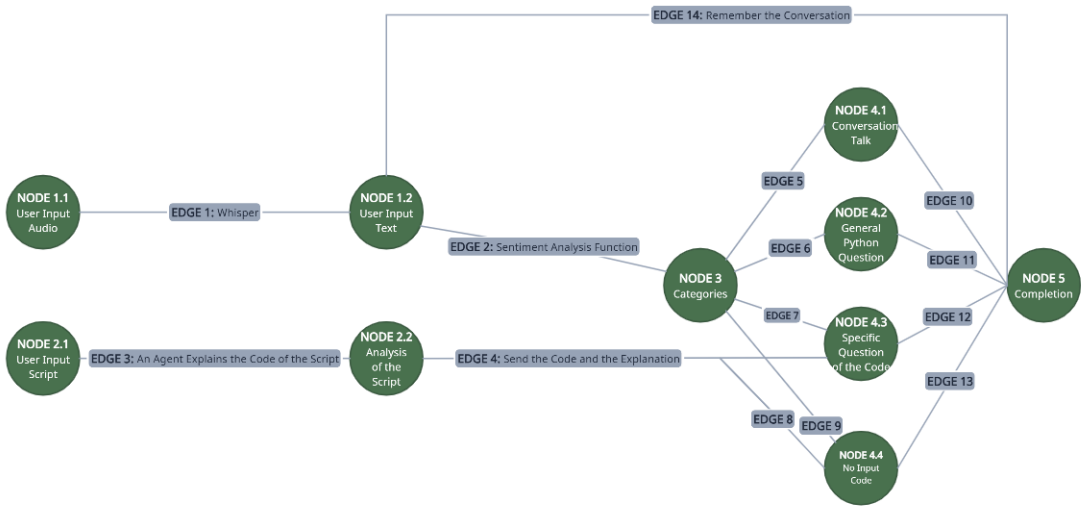


Figure 1: Flowchart of our App

First, we observe on the left side the nodes related to processing the user's input. Remember that this input can be in the form of audio (Node 1.1) or in the form of text (directly Node 1.2). When we receive an audio, we transcribe it using OpenAI's `whisper-1` model, obtaining a text string (arriving again at Node 1.2). The user also selects a file on which our model will operate (Node 2.1). This file is sent to a model trained to interpret and explain Python code. Therefore, an explanation is generated (Node 2.2) which, along with the code, will be provided to subsequent nodes in the future when needed.

Once we have the different user inputs are pre-processed, we perform a Sentiment Analysis (see Edge 2) powered by "gpt-3.5" to classify the user's request into one of the following categories:

1. The user wants to have a friendly conversation with the Chatbot, where, for example, they might ask what tasks it can perform.
2. The user has a general question about Python.
3. The user has specific questions about the code in their `.py` file.
4. The user presses the "send" button without typing or saying anything. This action will be interpreted as the user requesting a general description of the code in their file.

We observe that for categories 3 and 4 (corresponding to nodes 4.3 and 4.4), we will send the code and its previously generated explanation as an input, along with the user's text input. It is also worth noting that, depending on the identified category, different prompt engineering techniques will be applied to the user's input to achieve better results.

Finally, once we have precisely identified what the user wants, we can generate a response; in our case, using `gpt-3.5-turbo-instruct`. This response is then sent back to the firsts nodes of the graph to ensure the program remembers the entire conversation when generating future responses.

2.2 Training of the different Large Language Models

In this section we will briefly explain how the different LLMs used where trained, mainly, through prompt engineering.

Firstly, we initialized an array that will keep the conversation history with the following system-role instruction: *"You are an assistant whose task is to help the user programming in Python. You can only be asked questions about Python. If the user doesn't ask you anything about Python, just tell him who you are and ask him if he has any doubts, don't do anything alse. This applies when the user says you hello, asks who you are or simply asks anything not realted with Python. If the user does ask something about Python, you will act as an expert Python developer who is able to provide detailed solutions, advice and clear explanations of complex programming issues. Provide a code answering the question or solving the issue the user might have when needed. You should respond in a concise and professional manner. The format of your answer should be directly an answer, it must never try to finish the user query if it is unfinished".* The rest of the conversation of the ChatBot will be saved in a list similar to the {"role": , "content": } dictionary used by `OpenAI`, where the roles will be instead "human" and "ai".

2.2.1 Audio to text transcriber

To train the model in charge of the audio transcription we just send the prompt *"You are an assistant whose task is to help the user programming in Python. You will get an audio message in English. The output should be a transcription of the audio message also in English."* together with the audio file to the `whisper-1` model. Note that the audio version is only available in English; we will discuss language limitations in the future improvements section.

2.2.2 Code interpreter

We initialize the model `gpt-3.5-turbo-instruct` with a system instruction which says "*You are an expert Python developer. You are able to provide detailed solutions, advice and clear explanations of complex programming issues. You should respond in a concise and professional manner. You are talking with another expert, so use as much technical vocabulary as you want. The format of you're answer should be directly text explaining the given code*". This instruction is followed by an user-role instruction with the shape of the template "*Provide a short summary about the code: code. The format of the output should be: This code performs the following tasks <Explanation>*".

2.2.3 Sentiment Analysis

We give `gpt-3.5-turbo` the system instruction "*You are a helpful assistant whose role is to classify the input in different categories. The input will probably be something about programming.*" together with the user instruction "*Categorize the following text into one of the following categories: 1. Friendly chatting with the chat. The user asks the ChatBot about his job. If the input is really unclear, choose this category; 2. The user asks something general about Python, not specific about the user's code; 3. Specific doubts about the code in the script, the user explicitly references his code or talks about having an issue. The user can also ask if the issue is now correct, referring to modifications it just made on his code. Text: text. For example, for the text: <Who are you?>, the answer should be <1>, for the text <What is a for loop?> the answer should be <2> and for the text <Why my code does not work?> the answer should be <3>.*" Note that the fourth category does not need interpretation, as it is an empty string. The temperature of this model should be lowered to 0, as we want certainty in the answers.

From now on we will use `gpt-3.5-turbo-instruct` as the pre-trained model used to train the next LLMs.

2.2.4 Completion of Category 1.

We add to the conversation list the human-role instruction: "*Answer the following user query: {question}.*"", providing the user input as the `question` variable.

2.2.5 Completion of Category 2.

We add to the conversation list the human-role instruction: "*Respond to the question: {question}*"", providing the user input as the `question` variable.

2.2.6 Completion of Category 3.

We add to the conversation list the human-role instruction: "*We have the following code code. explanation. Answer the question about it: "question". Take into account what we have talked about the code previously. Always mention any change you make in the original code. The output should be given in the following format: <If and only if the user asks explicitly whether his code is correct, say whether the code is correct or not.> Explanation: <Short explanation of the code.> New version of the code: <The new version of the code>*", providing the user input as the `question` variable, the code of the script and the explanation of it previously generated as the `code` and `explanation` variables, respectively.

2.2.7 Completion of Category 4.

Remember that in this case there is no user input. Thus, we add to the conversation list the human-role instruction: "*Analyze and explain the following code: code. If and only if you think there is any issues or clear improvements, give a list of them. The improvements should focus on doing what the user wants to do, but better, not adding new functionality. The output should be given in the following format: Explanation: <The explanation of the code> Possible improvement:<A numbered list of the possible improvements>*", providing the code of the script as the `code` variable.

2.3 Design and Implementation of the App Interface

In this section we explain how we have created the interface used by the user to interact with the model explained previously. Our primary objective was to create a foreground window where the user can, firstly, specify the file they are working with, and secondly, chat in the most comfortable, quick, and direct manner possible.

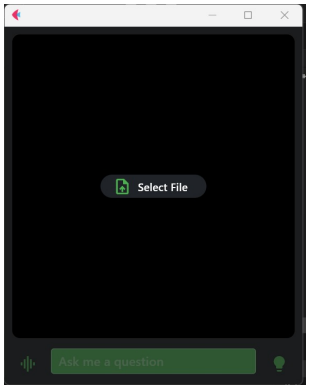
2.3.1 Stage 1: File Selection

The first part of the interface allows the user to select the file they are currently working on. This functionality is essential for users who need to reference or manipulate files directly from the chat interface. The File Picker Button is a button labeled "Select File" that, when clicked, opens a file dialog allowing the user to navigate their file system and select a file. See Figure 2a.

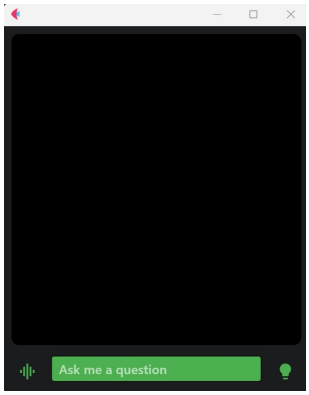
2.3.2 Stage 2: Chat Interface

The second part of the interface is the ChatBot system, designed enable communication between the user and the model. This interface needs to be intuitive and responsive, ensuring a fluent user experience. We can view it at Figure 2b.

Once the ChatBot in opened, we can see three different elements. Firstly, the Message Input Field is a text input field where users can type their queries. Then, the Send Button allows the user to send the query once is written or to ask for a code explanation when the is no input text. Finally, there is an audio button that allows the user to input its queries via audio.



(a) Screen where the user is able to upload a file.



(b) Empty ChatBot window.

The Chat Display Area is scrollable and allow users to see the whole conversation. Whenever there is a new message, the window should scroll to the latest message to keep the conversation flow uninterrupted. Each message are encapsulated in a Message Bubble. Messages from the user and responses from the ChatBot are visually distinct, each one disposed on each side of the interface.

3 Evaluation

3.1 Testing Methodology

To test the model, we will evaluate its performance in various scenarios that a programmer might encounter in their day-to-day activities. This will include testing how the model handles common programming tasks, debugging, code optimization, and answering general Python-related questions. By simulating real-world situations, we aim to ensure that the assistant is robust, reliable, and genuinely useful for programmers in their daily workflows.

3.2 Results

In this section, we will present some examples obtained using our application. These examples will demonstrate the assistant's capabilities, showcasing its effectiveness in providing code suggestions, debugging assistance, and answering Python-related questions. By including these examples, we aim to illustrate the practical utility and performance of our prototype.

Note that all the user inputs of this section can be always given via audio or written text. We want to emphasize that the transcription model worked perfectly in silent environments when the user talks in English.

Some Examples:

In the Figure 3 we can see how the model answers correctly two category 1 queries.

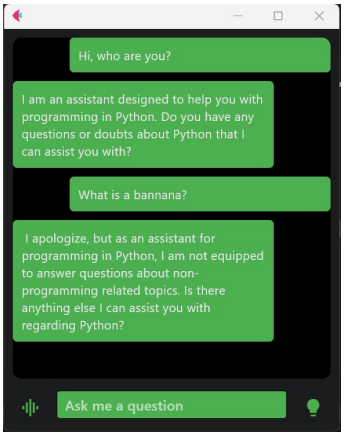


Figure 3: Example of the ChatBot answering to a category 1 query.

In the Figure 4 we can see how the model answers correctly a category 2 query.

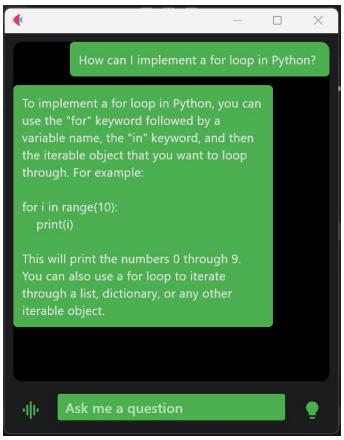


Figure 4: Example of the ChatBot answering to a category 2 query.

In the Figure 5 we can see how the model answers correctly a category 3 query.

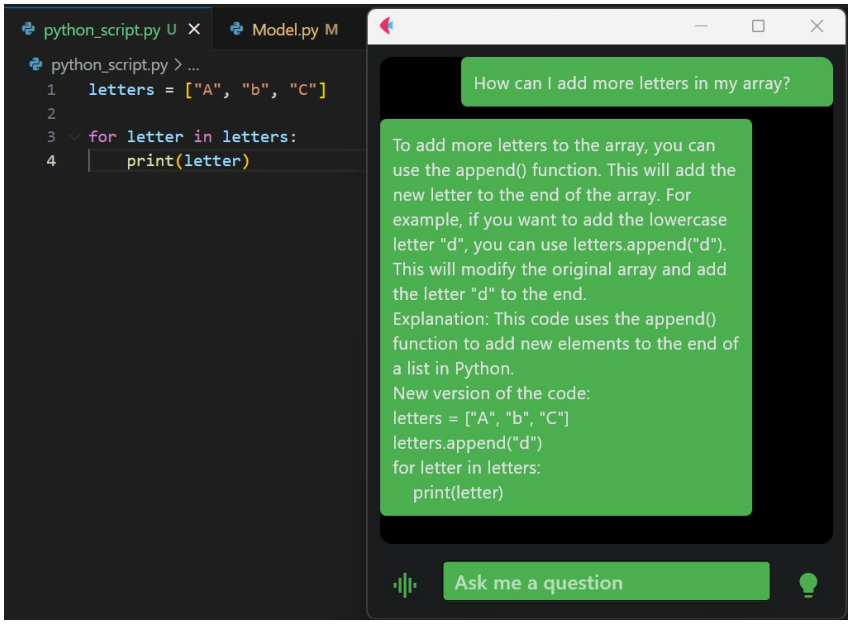


Figure 5: Example of the ChatBot answering to a category 3 query.

In the Figure 6 we can see how the model answers correctly a category 4 query.

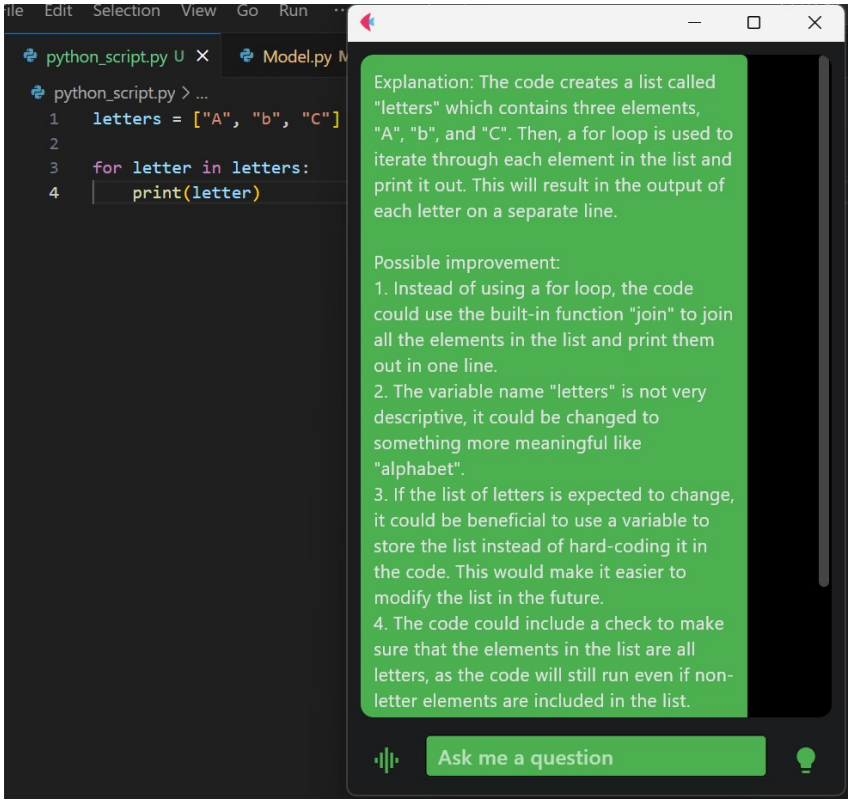


Figure 6: Example of the ChatBot answering to a category 4 query.

3.3 Discussion

The developed model is capable of efficiently and quickly performing simple tasks that a Python developer might need. However, when asked to perform more complex tasks, especially those that refer to the specific script the user is working on, it occasionally makes errors. This indicates that while the model is well-suited for basic assistance, there is still room for improvement in handling more intricate and context-specific requests. Additionally, the total cost of training and testing the model was just 28 cents, making it an affordable option.

3.4 Proposed Improvements

To close the evaluation section, we propose several improvements for the model. First, we aim to train the different models to perform better on complex user requests. This can be achieved by using few-shot examples in the prompts or even applying fine-tuning to the models. Additionally, we plan to implement *gpt-4o* instead of the other *gpt* models currently used. Although *gpt-4o* is a highly versatile model capable of performing all the tasks we need to carry out quickly and effectively; we were unable to implement it because it is so new that `Langchain` has not yet been updated to support it. Finally, we would also like to expand our horizons so that this assistant can work with different programming languages and handle programs distributed across multiple files. These improvements are expected to significantly enhance the assistant's performance and broaden its applicability.

4 Conclusion

In conclusion, we have successfully met our initial objectives, developing an application that efficiently assists Python developers with simple tasks. However, we acknowledge that there is significant room for improvement, particularly in enhancing the model's performance for handling more complex prompts. This is an area we intend to focus on in future iterations of the project, training the models with few-shot examples and even fine-tuning them.

We emphasize that the development process was constrained to a 24-hour period, during which we created the prototype, wrote the paper, and prepared the presentation. Given this tight timeframe, the progress we have made is commendable, yet it also highlights the potential for further refinement and optimization.

In summary, participating in this project has been an invaluable experience. It has provided us with a firsthand understanding of the capabilities of Generative AI in adding value to software development processes. Despite the challenges, we have demonstrated that AI can effectively support developers, and we are excited about the future improvements and applications of this technology.

References

[1] OpenAI. Openai's api documentation. https://platform.openai.com/docs/overview.

[2] DataCamp. Developing ai applications course on datacamp. https://app.datacamp.com/learn/skill-tracks/developing-ai-applications.